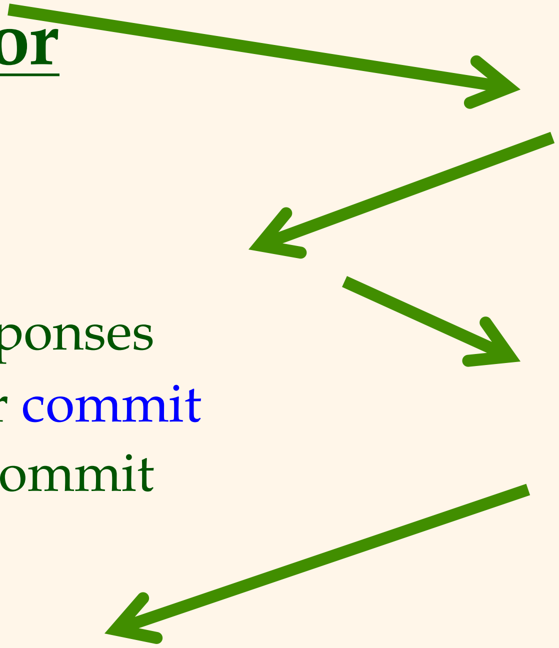
2PC (conclusion)

Last time: Two-Phase Commit



<u>Coordinator</u>

Send prepare

<u>Subordinate</u>

Make local decision

Send yes or no

Wait for all responses

Decide abort or commit

Send abort or commit

Perform abort or commit

Send ACK

Wait for all ACKs

State at coordinator

- Coordinator keeps some state (in memory) while running 2PC in a transaction table
- For each transaction
  - who are the subordinates
  - where we are in the protocol (which messages coordinator has sent/received)
- Ack messages from subordinates allow coordinator to garbage collect this state

2PC and failure

- Now let us talk about communication and site failures

2PC and comm failures

- If subordinate loses contact with a coordinator before receiving prepare?
  - Subordinate can decide to abort, since it hasn't voted so transaction cannot have committed

2PC and comm failures

- If coordinator loses contact with a subordinate before receiving all yes/no votes?
  - Coordinator can decide to abort since no-one has committed yet; must notify all subordinates who voted yes that we are aborting

2PC and comm failures

- If subordinate loses contact with a coordinator after voting but before receiving a commit/abort?
  - If voted no, can abort and is done
  - If voted yes, can't unilaterally decide what to do
  - Comm was lost during the uncertainty period

2PC and comm failures

- If subordinate loses contact with a coordinator after voting but before receiving a commit/abort?
  - Needs to communicate either with the coordinator, or possibly with another site to find out what the outcome was
  - Helps if coordinator tells each subordinate who the other subordinates are

2PC and comm failures

- If coordinator loses contact with subordinate after sending decision but before receiving ack?
  - Cannot garbage-collect transaction from coordinator state
  - When communication reestablished, can verify that subordinate knows about decision and then garbage-collect

2PC and site failures

- Now suppose the network is fine, but either the coordinator or a subordinate fails
- Need to remember sufficient state to allow recovery
- Will use a log for this
  - Coordinator and subordinate both log crucial steps in 2PC (and force log to disk so it survives a crash)
- A couple of variants for what is logged
  - Here we follow your course textbook on this
  - You may see minor differences elsewhere

Two-Phase Commit

Coordinator

Send prepare

Subordinate

Make local decision

Force-write prepare or abort

Send yes or no

Wait for all responses

Force-write abort or commit

Send abort or commit

Force-write abort or commit

Send ACK

Wait for all ACKs

Write (not force-write) end record

Restart after a failure

- Node crashes, comes back up
- Examines all in-progress 2PC transactions
  - Could be coordinator for some, subordinate for others
- Course of action based on last log record

- Desired behavior: if coordinator wrote commit to log, transaction is considered committed, else should abort

Restart after a failure

- Determine whether node was coordinator or subordinate
- Carry out recovery accordingly

Coordinator restart after a failure

- If have end log record, nothing to do
- If have commit or abort log record (but no end log record)
  - put transaction back into in-memory transaction table
  - know what the decision was; notify subordinates
  - wait for acks, clean up state and write end log record

Coordinator restart after a failure

- If don't have any log records
  - Can't have broadcast decision to subordinates
  - Decide to abort
  - If subordinates contact you asking for decision, can tell them it was abort
  - Could enter transaction back into transaction table, but no need
    - Default behavior: if you don't know anything about the transaction and a subordinate asks, tell them it was aborted.

Subordinate restart after a failure

- If have no (2PC-related) log entries, abort unilaterally
  - global decision can't have been a commit
- If have a commit or abort record, proceed accordingly

Subordinate restart after a failure

- If have prepare record but nothing else, cannot decide unilaterally
  - Site crashed in its uncertainty period
  - Needs to contact coordinator or other subordinates for what to do

2PC Optimizations

- Possible to optimize by reducing the number of messages and forced log entries in certain cases
- 2 optimizations:
  - Presumed Abort
  - Presumed Commit
- The XA standard for Distributed Transactions is 2PC with Presumed Abort

Two-Phase Commit

Coordinator

Send prepare

Subordinate

Make local decision

Force-write prepare or abort

Send yes or no

Wait for all responses

Force-write abort or commit

Send abort or commit

Force-write abort or commit

Send ACK

Wait for all ACKs

Write (not force-write) end record

Coordinator restart after a failure

- If have end log record, there is nothing to do
- If have commit or abort log record (but no end log record)
  - know what the decision was, proceed accordingly
- If don't have a commit or abort log record
  - decide to abort

2PC Presumed Abort

- If coordinator has no log records of transaction, it decides to abort
- So, if we decide to abort in a non-failure setting, can optimize by simply forgetting transaction (remove from transaction table)
  - No need to force-write abort log record at coordinator and subordinates
  - No need for acks from subordinates after abort
- But for commit, we proceed as in normal 2PC

2PC/PA, commit case

Coordinator

Send prepare

Subordinate

Make local decision

Force-write prepare

Send yes

Wait for all responses

Force-write commit

Send commit

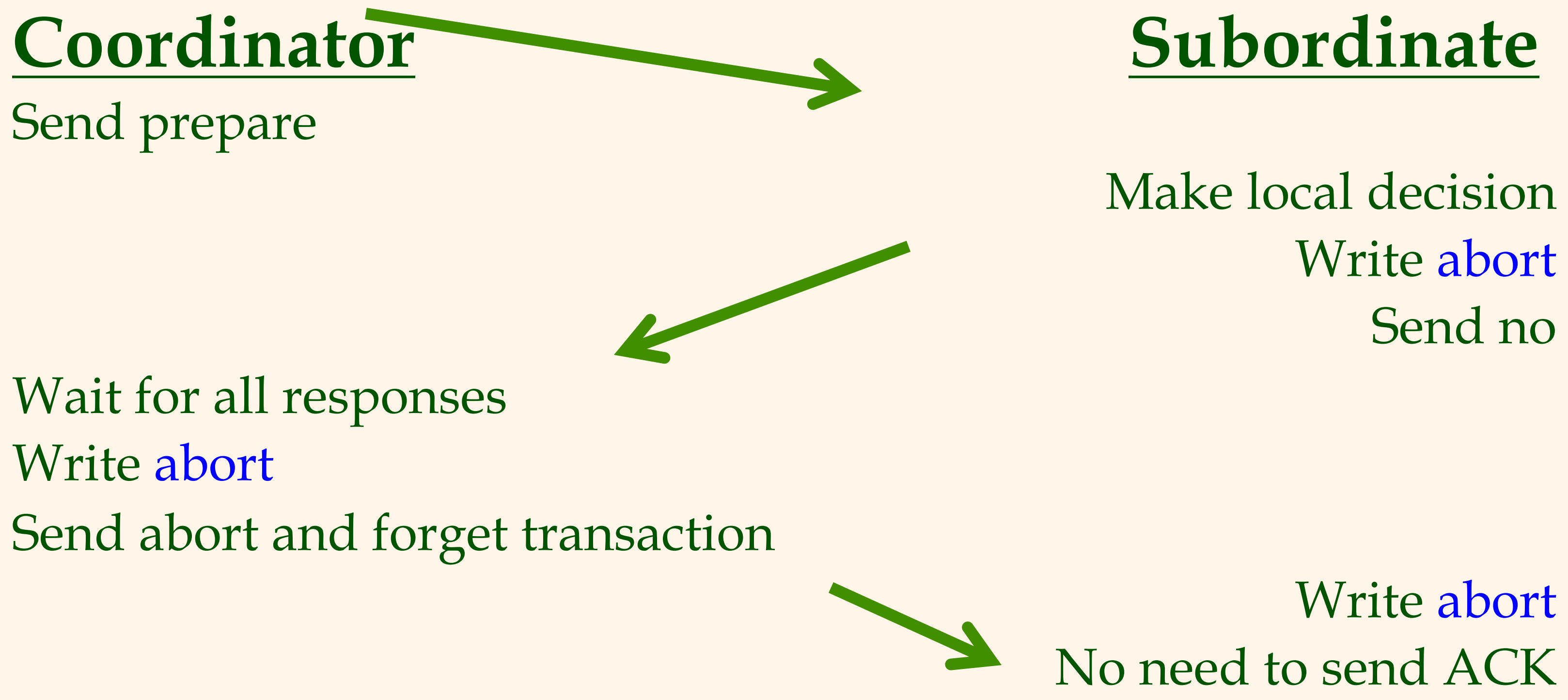
Force-write commit

Send ACK

Wait for all ACKs

Write (not force-write) end record

2PC/PA, abort case (subordinate voted no)



<u>Coordinator</u>

Send prepare

<u>Subordinate</u>

Make local decision

Write abort

Send no

Wait for all responses

Write abort

Send abort and forget transaction

Write abort

No need to send ACK

2PC Presumed Abort

- If subordinate crashes and queries coordinator on what to do, and coord. has garbage-collected transaction, will reply abort
- Of course, coordinator cannot garbage collect a committed transaction until it has received acks from all subordinates

Presumed Commit: Motivation

- Commit is the more common case! Let's optimize for it, not for abort
  - Require ack for ABORT not COMMIT
  - Subordinates force ABORT records, not COMMIT records
  - No information in transaction table: presume commit!

- We can do this, but the coordinator *must* force some extra records for correctness

2PC/PC, abort case

Coordinator

Send prepare

Subordinate

Make local decision

Force-write abort

Send no

Wait for all responses

Force-write abort

Send abort

Force-write abort

Send ACK

Wait for all ACKs

Write (not force-write) end record

2PC/PC, commit case, first try

Coordinator

Send prepare

Subordinate

Make local decision

Write prepare

Send yes or no

Wait for all responses

Write commit

Send commit and forget state

Write commit

No need to send ACK

Presumed Commit

- Suppose coordinator crashes and comes back up; needs to figure out what to do
- If there are no log records, does it mean that it decided to commit before crash?
  - Or does it mean that it only sent PREPARES and didn't decide to commit yet?
  - Need to be able to distinguish between the two because actions to be taken are different!!

The solution

- Coordinator force-writes begin/prepare record (upon start of protocol) AND commit record (upon decision to commit)
- Subordinates do not need to force commit log records
- Now after crash recovery, either:
  - Coord has begin but no commit -> rollback
  - Coord has begin and commit -> commit

2PC/PC, commit case

Coordinator

Send prepare

Force-write begin

Subordinate

Make local decision

Write prepare

Send yes or no

Coordinator

Wait for all responses

Force-write commit

Send commit and forget state

Subordinate

Write commit

No need to send ACK

Additional 2PC Optimization

- Subordinates who only read send READ votes instead of YES votes
  - No log writes!
- Coordinator logic
  - READ & YES = YES
  - READ & NO = NO
  - READ & READ = READ
- If READ at coordinator, no need for second phase! Else, only contact non-READs.

2PC summary

- Basic version
- Handling comm and site failures
  - A subordinate cannot always unilaterally recover
  - If failure occurred during its uncertainty period
- Optimizations to reduce messages, logging
  - Presumed Abort/Commit
  - Special treatment of readers

2PC and blocking

- 2PC is a protocol that may block even when a portion of the nodes are up (non-total failure)
- Blocks if a subordinate is in its uncertainty period and can only contact other subordinates who are in their uncertainty period
  - Could block indefinitely until they can finally reach someone who knows what to do
    - Coordinator or a subordinate not in its uncertainty period

Blocking protocols

- There are theoretical limitations on our ability to avoid blocking in a commit protocol
  - (while still retaining correctness)
- For an in-depth discussion see Phil Bernstein's textbook, Chapter 7

Blocking protocols

- There are protocols which reduce the probability of blocking
- Example: 3PC (Three-phase commit)
- If no comm failures, 3PC will not block as long as a majority of sites are operational
  - Think about why 2PC does not guarantee that!

Three Phase Commit

- Phase 1: Voting as before
- Phase 2: Dissemination of results
  - If coordinator gets all "yes" votes, sends "precommit" message
  - When coordinator gets acks from a majority of the sites, actually makes decision to commit
- Phase 3: Termination as before

3PC

- Reduces chance of blocking
- Phase 2 makes sure that the decision to commit is recorded on a majority of sites before the final order to commit is issued

3PC

- Recovery: no comm failures, majority of sites are up
- If no-one has a "precommit" message, coordinator cannot have issued final order to commit
  - safe to abort
- If someone has "precommit" message, knows decision was going to be commit
  - So safe to commit!